Using Markov Chain and Graph Theory Concepts to Analyze Behavior in Complex Distributed Systems

Christopher Dabrowski
U.S. National Institute of Standards and Technology
Gaithersburg, MD, USA
cdabrowski@nist.gov

Fern Hunt
U.S. National Institute of Standards and Technology
Gaithersburg, MD, USA
fhunt@nist.gov

***Abstract*— We describe how Markov chain analysis and graph theory concepts can be used together to efficiently analyze behavior of complex distributed systems. Specifically, the paper shows how minimal s-t cut sets can be used to identify state transitions in a graph of a piece-wise homogenous Markov chain, which if suitably perturbed, degrade the performance degradations in the system being modeled. Using a computing grid simulation as an example, the paper shows that this approach can be used to predict situations where performance degrades under different conditions. Preliminary experiments show this approach can be applied to problems of significant size. We conclude that the use of minimal s-t cut-sets to perturb a Markov chain has potential for practical use in analyzing distributed systems behavior and describe how this might be done in support of real-time system monitoring. This approach has not previously been studied as a practical means for analyzing dynamic behavior in distributed systems.**

Keywords- piece-wise homogenous Discrete Time Markov chain; distributed system; minimal s-t cut set; system monitoring.

I. INTRODUCTION

In large-scale, dynamic distributed systems, such as computing grids, the interactions of many independent components can lead to emergent system-wide behaviors with unforeseen, often detrimental, outcomes [1]. To ensure availability and reliability of computing services in such environments, system monitoring tools will be needed to rapidly assess trends and predict changes in system behavior caused by such factors as shifts in workload, modifications to system configurations, policy changes, or failures.

In an earlier paper [2], we described a succinct Discrete Time Markov chain (DTMC) representation for analyzing dynamic behavior in distributed systems. To capture change in system behavior over time, the DTMC representation was made *piecewise homogenous* [3], in which a set of transition probability matrices (TPMs) modeled successive time periods. This differs from a conventional DTMC that is homogeneous, in which a TPM does not change over time. The TPM set could be perturbed by systematically changing the values of related state transition probabilities to examine alternative system execution paths and identify *critical state transitions* where perturbation most affected performance. The critical transitions could then be related to events such as failures, policy changes, and workload shifts, in order to predict the extent of performance decline caused by such events and to establish related performance thresholds. This initial approach however, required extensive computation, which slowed analysis.

This paper extends [2] to use minimal s-t cut-sets [4] of a graph of a piecewise homogenous DTMC to rapidly identify critical state transitions, which when suitably perturbed, lead to large performance degradations. Use of minimal s-t cut sets, defined in Section IV, reduces the computation needed to find critical transitions, and can thus be applied to more complex problems than our initial approach. We show minimal s-t cut analysis can be efficiently used in grid system simulations with different durations and workloads, and that it can also find combinations of critical transitions that represent more complicated circumstances. For larger problems, we provide an algorithm for finding minimal s-t cut sets that can be bounded to run within reasonable time limits, and that is effective in identifying critical transitions in DTMCs with up to 50 states and 160 state transitions. We then discuss the potential use of these capabilities to predict performance degradations in support of real-time system monitoring. To our knowledge, the use of minimal s-t cut-set analysis to perturb a piecewise homogenous DTMC has not previously been studied as a practical approach for analysis of dynamic behavior in distributed systems.

This paper is organized as follows. Section II discusses related work on using Markov chains to analyze dynamic systems. Section III summarizes the DTMC for the grid system [2] used here. Section IV defines minimal s-t cut sets and describes how they are used to find critical transitions in a DTMC. Section V describes an algorithm for computing minimal s-t cut sets in large Markov chain graphs and analyzes its performance. Section VI discusses potential use of minimal s-t cut set analysis to monitor real-time systems. The last sections outline future work and conclude.

II. RELATED WORK

The method discussed here should be distinguished from the well-known use of DTMCs for providing quantitative measures of system performance and reliability, reviewed in [2, 5]. Of this work, perhaps most closely related are [6, 7], where a control loop is employed to mitigate network delay and a Markov chain model is used to represent and measure

delay. Instead of measuring reliability, we use DTMCs to examine alternative execution paths in dynamic systems and identify scenarios where performance degrades.

Both perturbation analysis and graph theory have previously been applied to DTMCs, but for different purposes than we intend. Perturbation analysis of DTMCs has been the topic of theoretical [8-10] and computational study [11, 12]. Other researchers [13-15] have used system performance gradients that are based on key decision parameters to perturb Markov models. While gradient-based approaches demonstrated potential in modeling performance change, some issues involving computation of gradients required further research [15]. Also, gradient-based approaches seem geared for system optimization, rather than for examining alternative execution paths and identifying situations in which performance degrades.

Graph-theoretic methods have also been used previously to study dynamic behavior in Markov chain models. For example, graph decomposition has been used to calculate stationary probability distribution vectors of Markov chains [16-18] as well as to measure how perturbation affects stationary distributions [19]. Minimal cut set analysis has been used on topology graphs of avionics system components to identify the shortest sequence of component failures [20]. However, the combined use of minimal s-t cut sets and a piecewise homogenous DTMC representation has not previously been studied as an approach for analysis of distributed systems behavior. Finally, there are maximum-flow algorithms [21-23], well-known graph-theoretic methods that find s-t cut sets on the basis of flow levels. These algorithms could potentially be used to identify critical state transitions. However, because these algorithms use flows, they are distinguishable from Markov chain approaches and so best merit separate investigation.

III. The Discrete Time Markov Chain Model

The DTMC model of a grid system was developed by observing a large-scale grid computing simulation [1]. This section overviews this model, with full details in [2]. The DTMC grid model simulates the progress of a large number of computing tasks from the time they are submitted to the grid for execution to the time they either complete or fail. Fig. 1 shows a state diagram of this system, which describes the lifecycle of a single task. This model has 7 states: an *Initial state*, where a task remains prior to submission; a *Discovering* state, during which service discovery directories are accessed to locate grid service providers to execute the task; a *Negotiating* state during which a Service Level Agreement for task execution is negotiated with discovered providers; a *Waiting* state where tasks reside that are temporarily unsuccessful in discovery or negotiation; and a *Monitoring* phase in which a task is executed by a contracted provider before a deadline. Transitions between states, shown in Fig. 1 by arrows, represent actions taken by the grid system to process a task. All tasks eventually enter either the *Tasks Completed* or *Tasks Failed* state, which are the *absorbing states* of the Markov chain, because once entered, a task cannot leave. A Markov chain with these characteristics is called an *absorbing chain*.

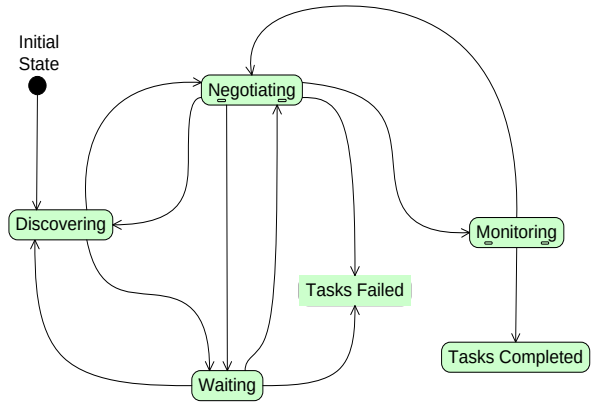


Figure 1. State model of grid computing system.

The large-scale grid simulation was observed over extended durations to accumulate frequencies for the state transitions shown in Fig. 1, compute transition probabilities, and form TPMs. Separate observations were made to create two cases: (a) one in which the system executes for 8 hours with varying workload; and (b) a 640 hour execution that reaches near steady state. To create *piece-wise homogenous* representations for the two cases, the total duration of each was divided into equal periods of 7200 s and a TPM was computed for each period. Fig. 2 shows the summary TPMs for the two cases, which are weighted averages of their respective time period TPMs and in which weights are based on the relative number of transitions in each period.

(a)

	Initial	Wait	Disc	Ngt	Mon	Comp	Fail
Initial	0.9697	0	0.0303	0	0	0	0
Waiting	0	0.8363	0.0673	0.0918	0	0	0.0046
Disc	0	0.0355	0.6714	0.2931	0	0	0
Ngt	0	0.4974	0.0182	0.2882	0.1961	0	0.0001
Mon	0	0	0	0.0003	0.9917	0.0080	0
Comp	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

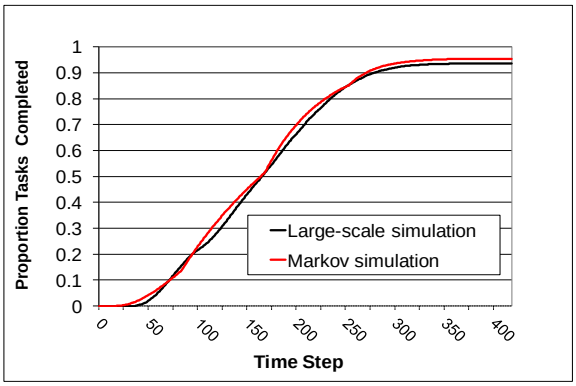
(b)

	Initial	Wait	Disc	Ngt	Mon	Compl	Fail
Initial	0.9997	0	0.0003	0	0	0	0
Wait	0	0.7612	0.0460	0.1911	0	0	0.0017
Disc	0	0.0686	0.6084	0.3230	0	0	0
Ngt	0	0.2401	0.0062	0.2378	0.4801	0	0.0358
Mon	0	0	0	0.0007	0.9902	0.009	0
Compl	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

Figure 2. (a, b) Summary TPMs for the grid system over (a) 8 and (b) 640 hour durations. The summary TPMs are weighted averages of their component time period TPMs, in which the weight of each TPM is determined by the relative number of transitions in the related time period.

To simulate system behavior over time, a well-known DTMC method was employed in which multiplication of the time period TPMs is used to advance the system state in discrete time steps. Fig. 3 shows that this process, referred to in this paper as *Markov simulation*, closely approximates the performance of a large-scale simulation in the two cases.

(a)



(b)

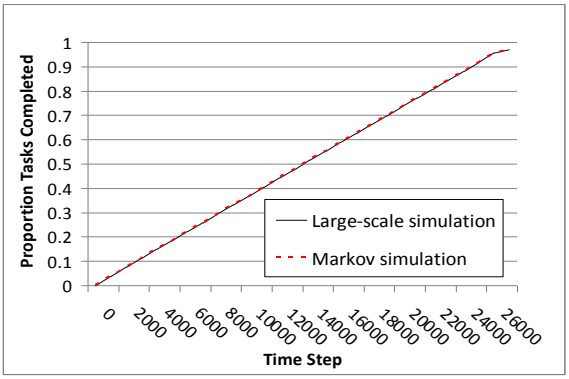


Figure 3. Performance of Markov chain and large-scale simulations as measured by Tasks Completed for the grid system over (a) 8 hours (5 time period TPMs and 421 time steps, including an extra cleanup period), and (b) 640 hours (steps 321 time period TPMs and 27000 time steps with cleanup). A time period represents 7200s and a time step represents 85 s.

To identify critical state transitions, we described a perturbation algorithm in [2], which systematically changes values of combinations of related state transition probabilities in a piecewise homogenous DTMC and evaluates alternative system executions using Markov simulation. The algorithm identifies those combinations that degrade system performance. The algorithm can be applied to all rows (states) to exhaustively perturb a TPM and reveal individual critical state transitions where perturbation causes system performance to fall. The algorithm replicates (with good agreement) scenarios in which performance drastically degrades in a large-scale grid simulation [1].

Fig. 4 provides an example of a critical state transition, *Negotiating* → *Monitoring*, identified by the perturbation algorithm. The figure shows the impact of a set of related perturbations in which lowering the probability of transition to 0 for *Negotiating* → *Monitoring* causes the proportion of *Tasks Completed* to fall to 0 in the Markov simulation (blue curves). The figure also shows the result of the large-scale grid simulation (red curve), which was altered to deliberately fail negotiations. The Markov simulation curve shows a similar threshold and sharp drop in *Tasks Completed* as does the large-scale simulation. Though the computational cost of the perturbation algorithm prohibits use on large problems, it provides a comparative basis to assess the results of minimal s-t cut-set analysis.

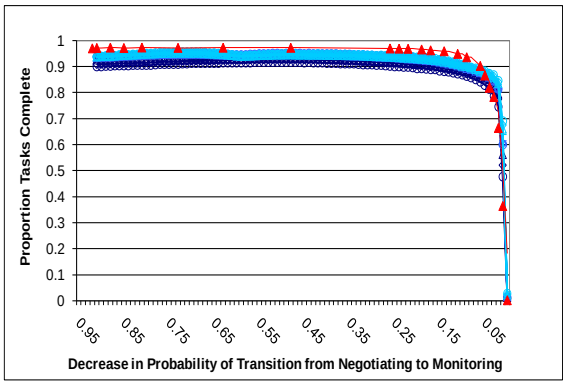


Figure 4. Perturbation of *Negotiating* State to reduce the probability of transition from *Negotiating* → *Monitoring* while raising the probability of transition from *Negotiating* to *Waiting* in the 640 hour case. The blue curve shows the proportion of *Tasks Completed* produced by the perturbation algorithm. Large-scale simulation results are denoted by red triangles.

IV. MINIMAL S-T CUT SET ANALYSIS IN A MARKOV CHAIN MODEL

This section describes how identifying minimal s-t cut sets on paths between an *Initial* state and a desired absorbing state can be used to identify critical state transitions in a DTMC, which if perturbed, lead to system performance degradations. In contrast to the perturbation algorithm, which can identify only single state transitions that are critical, minimal s-t cut set analysis identifies combinations of critical state transitions, an important benefit for analysis of more complex problems. In Section V, we describe an algorithm for finding minimal s-t cut sets and show its effectiveness for large problems. We note that this approach does not use flow levels to define minimality, but instead uses cardinality and other factors discussed below

A. *Definitions*

In graph theory, a graph G (V, E) consists of a set of vertices V connected by edges from the set E. A directed graph is a graph in which edges can be traversed in only one direction. It is easy to see that a Markov chain is a directed graph, in which vertices correspond to states and directed edges correspond to state transitions. A directed path through this graph is a sequence of state transitions from one state to another. In this problem, the directed paths of most interest are non-cyclic paths that lead from the *Initial* state to one of the two absorbing states: *Tasks Completed* or *Tasks Failed*. This paper considers only paths to *Tasks Completed*. Fig. 5 shows two such paths.

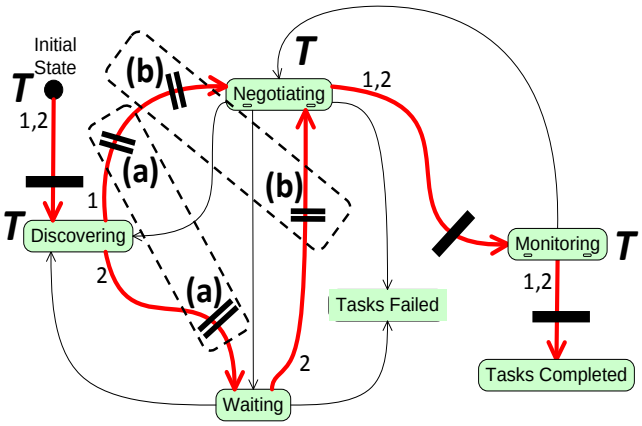


Figure 5. There are 2 directed paths (in red) from the *Initial State* to *Tasks Completed*, labeled 1 and 2. Three single-transition s-t cuts (minimal s-t cut sets consiting of one state transition) are marked by single bars. Two multiple transition s-t cuts (s-t cut sets with two transitions each) are marked by double bars: (a) *Discovering* → *Negotiating* and *Discovering* → *Waiting*; and (b) *Discovering* → *Negotiating* and *Waiting* → *Negotiating*. Trap states are denoted by ***T***.

A set of one or more edges, which if removed, disconnects all paths between two vertices *s* and *t* is referred to as an *s-t cut set* [4]. An s-t cut set is a *minimal s-t cut set* if removal of any edge from the cut set reconnects *s* and *t*. By finding minimal s-t cut sets consisting of state transitions that disconnect the *Initial* and *Tasks Completed* states, it is possible to know where reducing the related transition probabilities to 0 can halt the progress of tasks to completion. In this paper, minimal s-t cut sets with a single member will be referred to as *single-transition s-t cuts*, while those with more than one member are *multiple-transition s-t cuts*. State transitions that are members of a minimal s-t cut set are critical state transitions.

B. *Identifying Minimal s-t Cut Sets in the Grid Markov Chain Model*

In Fig. 5, there are 3 single-transition s-t cuts: *Initial* → *Discovering*, *Negotiating* → *Monitoring*, and *Monitoring* → *Tasks Completed*. Fig. 4 has shown that reducing the probability of transition for *Negotiating* → *Monitoring* to 0 using Markov simulation causes the proportion of tasks reaching *Tasks Completed* to drop to 0. The same result occurs when the other two single-transition s-t cuts, *Initial* → *Discovering* and *Monitoring* → *Tasks Completed*, are similarly perturbed (see [2]). Exhaustive use of the perturbation algorithm confirms that the 3 single-transition s-t cuts identify state transitions, which if reduced to 0, cause the proportion of tasks reaching the *Tasks Completed* state to fall to 0 (see Sec. V). These 3 single-transition s-t cuts are critical state transitions that are clearly related to critical steps in allocating resources to, and executing, tasks. Fig. 5 also shows two multiple-transition s-t cuts, labeled (a) and (b), which disconnect all paths between the *Initial* from the *Tasks Completed* state. Both multiple transition cuts have two transitions. In a multiple-transition s-t cut, lowering transition probabilities to 0 of all transitions in the cut set reduces the proportion of *Tasks Completed* to 0. Multiple-transition s-t cuts identify situations where a combination of state transitions is critical and together points to circumstances that degrade system performance. We return to multiple-transition s-t cuts in Section V.

C. *Identifying Trap States*

The previous discussion considered only state transitions between different states. However, in a DTMC, a state may also transition to itself in the next discrete time step and remain in the same state. In this paper, this is referred to as a *self-transition*. If a self-transition probability is near 1, the task may stay in the state for a long time. Such a state effectively becomes a *trap state*. Fig. 6 shows an example of how a *trap state* affects performance, when the self-transition probability of the *Discovering* state is raised to 1.

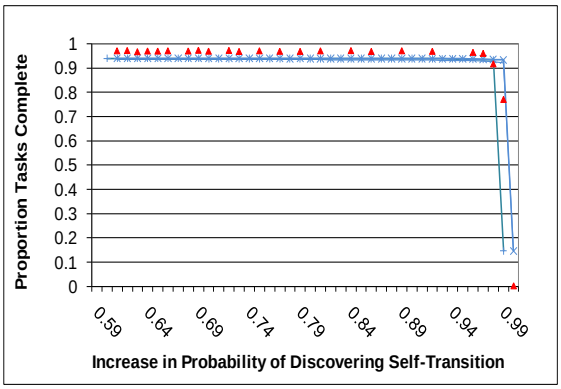


Figure 6. Perturbation of *Discovering* to increase the self-transition probability to 1 while decreasing the probability of transition from *Discovering* to other states to 0 in 640 hour case. The blue curve shows the proportion of *Tasks Completed* produced by the perturbation algorithm. Values from the large-scale simulation are shown by red triangles.

Tasks are thus stalled in *Discovering*, so that they cannot proceed to other states and complete by their deadlines. The evolution of *Discovering* into a trap state may indicate failure of the service discovery function in a real-world scenario. A trap states is distinguished from a permanent absorbing state, which always has a self-transition probability of 1, while the self-transition probability of a trap state varies. The concept of an s-t cut set can be extended to include vertices whose removal cuts all paths from *s* to *t*. A minimal set of such elements (edges and vertices) is a *minimal s-t separating set* [24]. We leave minimal s-t separating sets for potential future work.

V. Performance of Minimal s-t Cut Set Analysis

This section shows that minimal s-t cut set analysis is an effective and efficient means of finding critical state transitions and trap states. The section also shows the potential applicability of this approach to large problems. Section V(*A*) first describes a well-known algorithm, known as the *node contraction algorithm*, which we adapted to find

minimal s-t cut sets in a directed graph. The node contraction algorithm is probabilistic [25], in that can find solutions with a probability which can be increased to a high level by repeatedly executing the algorithm. Though it is not guaranteed to find all minimal s-t cut sets, the computational cost of node contraction can be bounded, making it potentially applicable to large problems where the perturbation algorithm and other exhaustive methods would be infeasible. The node contraction algorithm also finds critical transitions that are part of multiple transition s-t cuts, which the perturbation algorithm cannot find.

Section V(*B*) shows effectiveness and efficiency of node contraction for the grid system problem. Here, the node contraction algorithm finds all individual critical transitions and trap states that were found using the perturbation algorithm, but at far less cost. Section V(*C*) then reports experiments on the use of node contraction for finding critical state transitions in large Markov chain problems. To do this, the performance of the node contraction algorithm is tested by comparing it to the performance of an algorithm [26] that, unlike node contraction, enumerates *all* minimal s-t cut sets in a directed graph, and thus finds all critical transitions. While, like other algorithms of this type, the time complexity of [26] prohibits practical use on many large problems, the algorithm provides a good baseline for testing. (This complexity is $O\ |E|$ for each s-t cut set that exists, where $|E|$ is the number of edges in the graph [26]). To examine the potential for scalability of minimal s-t cut set analysis, we wish to know what proportion of minimal s-t cut sets (and thus critical transitions) can be found by node contraction in large problems and its computational cost.

A. Overview of the Node Contraction Algorithm

In this section, we summarize our implementation of the node contraction algorithm, with pseudo-code in [5]. Though the time complexity of node contraction algorithms for directed graphs has not been studied, efficient versions of this algorithm for undirected graphs find a minimum cut with a complexity of $O\ |V|^2]$, where $|V|$ is the number of vertices in the graph [25]. While this cost is significant, the algorithm can be used on large problems by controlling the number of executions, as will be discussed in Section V(*C*).

The node contraction algorithm operates by randomly choosing two vertices connected by an edge and replacing these vertices with a single, new vertex. The new vertex assumes the edges by which the two replaced vertices were connected to the remainder of the graph (i.e., the edges of replaced vertices become the edges of the new vertex) and takes up the edges that connected the two replaced vertices. The result of each contraction is recorded. The process of randomly selecting pairs of vertices repeats until only two large, *mega-vertices* remain. The directed edges between the two remaining mega-vertices c_1 and c_2, and the directed edges between vertices $<v_1, v_2>$, $v_1 \neq v_2$, in which v_1 was replaced by c_1 and v_2 was replaced by c_2, constitute a minimal s-t cut set of the graph. The node contraction algorithm was modified for an absorbing Markov chain problem to prevent the two vertices representing the *Initial* state, *s*, and the *Tasks Completed* absorbing state, *t*, from being contracted into the same vertex. This ensures that the *Initial* state, *s*, and *Tasks Completed* state, *t*, will not both end up in either c_1 or c_2. In this way, the edges between the two remaining mega-vertices, c_1 and c_2, together with the vertices each has absorbed, yield a minimal s-t cut set of state transitions, which if removed, disconnect the *Initial* and absorbing state (*Tasks Completed*).

Since the algorithm randomly selects two connected vertices to combine, repeated applications produce different cut sets. The more the algorithm is repeated, the greater the chances that a large proportion, if not all, of the minimal s-t cut sets of interest will be obtained. Hence, the operation of the algorithm can be said to be probabilistic. Because the number of repetitions can be controlled, computation cost can be bounded. Further, cut sets can identify potential trap states, which exist when all transitions in the cut set emanate from the same state. Markov simulation then need be applied only to the transitions in the s-t cut sets, in order to generate curves of tasks completed, such as are shown in Figs. 4 and 6, and to identify performance thresholds.

B. Comparing the Perturbation AlgorithmWith Minimal s-t Cut Set Analysis Using Node Contraction

Table I compares the result of applying the perturbation algorithm described in Section III with the result of minimal s-t cut set analysis using node contraction, when both are used to identify individual critical state transitions and trap states. The perturbation algorithm was applied to the 5 rows representing non-absorbing states (labeled a-e) for each time period TPM in both the 8 and 640 hour cases. The combinations of row elements representing the probability of transition being decreased and increased appear in the 2 leftmost columns. For each such combination of transitions, the next two columns show the proportion of *Tasks Completed* for the 8 and 640 hour cases as the probability of the state transition being decreased falls to 0. The rightmost column indicates if the state transition being reduced corresponds to a single-transition s-t cut.

Table I shows that all combinations where perturbation caused a decline in the proportion in *Tasks Completed* that fell to 0 corresponded to single-transition s-t cuts. There are 7 such combinations, and all correspond to single-transition s-t cuts that are verified by large-scale simulation. In no case, did node contraction find an s-t cut that did not correspond to such a drastic reduction. For instance, in Table I(c), rows 10-12, when the probability of transition from the *Negotiating* state to *Monitoring* (i.e. *Negotiating* → *Monitoring*) is reduced to 0, the proportion of *Tasks Completed* falls to 0. This is shown in Fig. 4. Fig. 5 shows that *Negotiating* → *Monitoring* is a single-transition s-t cut.

TABLE I. Comparison of results of applying the perturbation algorithm to completely perturb all rows of the TPMs for 8 and 640 hour cases and existing single-transition s-t cuts found by the node contraction algorithm. Perturbations verified by altering the large-scale grid system simultion are bolded and shaded.

(a) *row* = *Discovering*

	Element reduced→0	Element raised	Proportion of *Tasks Complete* 8-hour	Proportion of *Tasks Complete* 640-hour	s-t cut exists
1	*Waiting*	*Discovering*	0.957	0.935	No
2	*Waiting*	*Negotiating*	0.959	0.935	No
3	*Discovering*	*Waiting*	0.939	0.935	No
4	*Discovering*	*Negotiating*	0.963	0.935	No
5	*Negotiating*	*Waiting*	0.894	0.933	No
6	*Negotiating*	*Discovering*	0.651	0.932	No

(b) *row* = *Waiting*

	Element reduced→0	Element raised	Proportion of *Tasks Complete* 8-hour	Proportion of *Tasks Complete* 640-hour	s-t cut exists
1	*Waiting*	*Discovering*	0.974	0.937	No
2	*Waiting*	*Negotiating*	0.981	0.939	No
3	*Discovering*	*Waiting*	0.937	0.934	No
4	*Discovering*	*Negotiating*	0.963	0.936	No
5	*Negotiating*	*Waiting*	0.818	0.843	No
6	*Negotiating*	*Discovering*	0.939	0.932	No

(c) *r* = *Negotiating*

1	*Waiting*	*Discovering*	0.974	0.937	No
2	*Waiting*	*Negotiating*	0.985	0.938	No
3	*Waiting*	*Monitoring*	1.000	0.939	No
4	*Discovering*	*Waiting*	0.954	0.935	No
5	*Discovering*	*Negotiating*	0.957	0.935	No
6	*Discovering*	*Monitoring*	0.967	0.936	No
7	*Negotiating*	*Waiting*	0.923	0.931	No
8	*Negotiating*	*Discovering*	0.941	0.933	No
9	*Negotiating*	*Monitoring*	0.988	0.938	No
10	*Monitoring*	*Waiting*	***0.000***	***0.000***	**Yes[a]**
11	*Monitoring*	*Discovering*	***0.000***	***0.000***	**Yes[a]**
12	*Monitoring*	*Negotiating*	***0.000***	***0.000***	**Yes[a]**

(d) *row* = *Monitoring*

1	*Negotiating*	*Monitoring*	0.982	0.937	No
2	*Negotiating*	*Tasks Comp*	0.982	0.938	No
3	*Monitoring*	*Negotiating*	***0.028***	***0.186***	**Yes[b]***
4	*Monitoring*	*Tasks Comp*	0.980	0.949	No
5	*Tasks Comp*	*Negotiating*	***0.001***	***0.006***	**Yes[b]**
6	*Tasks Comp*	*Monitoring*	***0.002***	***0.016***	**Yes[b]**

(e) *row* = *Initial*

1	*Discovering*	*Initial*	***0***	***0***	**Yes[c]**
2	*Initial*	*Discovering*	0.970	0.988	No

In Fig. 5, corresponds to single-transition s-t cuts for
[a]*Negotiating* to *Monitoring*
[b]*Monitoring* to *Tasks Completed*
[c]*Initial* to Discovering

*Note explanation in text on this page below for (d), row 5.

Note that in Table I(d), row 3, reducing the probability of *Monitoring* self-transition while raising the probability of *Monitoring* → *Negotiating* also caused a severe decline in the proportion of *Tasks Completed*. This happens because the probability of transition for *Monitoring* → *Tasks Completed* is very low (see Fig. 2), and so the probability of *Monitoring* self-transition must be very high to ensure tasks remain in the *Monitoring* state long enough to eventually transition to *Tasks Completed*. Thus, reducing the probability of *Monitoring* self-transition to 0 while raising the transition probability of *Monitoring* → *Negotiating* prevents tasks from reaching *Tasks Completed*—and acts like a single-transition s-t cut on *Monitoring* → *Tasks Completed*. However, because the transition probability of *Monitoring* → *Tasks Completed* is not lowered to 0 in this perturbation, some tasks complete. The table also contains rows that show only partial reductions (Table I (a), rows 5-6, and Table I (b), row 5). These correspond to the state transitions in the two multiple transition s-t cuts in Fig. 5, which were identified by node contraction, but could not be identified by the perturbation algorithm of Section III.

The perturbation algorithm was also applied to raise the self-transition probability of the 5 non-absorbing states in the grid model to 1. This perturbation caused the proportion of *Tasks Completed* to decline to 0 when applied to 4 of these states: *Initial*, *Discovering*, *Negotiating*, and *Monitoring* states. All 4 are trap states found through node contraction. The fifth state, *Waiting*, is not a trap state; but is part of a state transition that is a member of the two multiple-transition s-t cuts in Fig. 5. Hence, if the self-transition probability of *Waiting* is raised to 1, there is only a partial downward effect on the proportion of task completions.

Executing the exhaustive perturbation algorithm on non-absorbing rows of the grid model took 56 minutes in the 8 hour case and 4.5 hours in the 640 hour case. In comparison, node contraction needed less than 0.01 s to find all minimal s-t cut sets and trap states. In the 8 hour case, generating Markov simulation curves to reduce the proportion of *Tasks Completed* to 0 for the minimal s-t cut sets and trap states required 244 s, or 7 % of the 56 minutes needed by the perturbation algorithm. In the 640 hour case, these same computations took 230 s, or 1.4 % of the 4.5 hours needed by the perturbation algorithm. Thus, minimal s-t cut set analysis yielded two orders of magnitude improvement in run time over exhaustive application of the perturbation algorithm. All experiments were executed on a Dell PowerEdge 6950 with quad, dual-core 3.0GHz processors and 32GB memory, running under Windows 2003.

C. *Using Node Contraction to Find Minimal s-t Cut Sets in Large Problems*

This section reports the results of experiments on the use of the node contraction algorithm to find critical transitions in large, complex Markov chain models with many multiple transition minimal s-t cuts. These experiments compare the results of using the contraction algorithm to the results produced by the enumeration algorithm of [26] which is guaranteed to find all minimal s-t cut sets and the critical transitions that these cut sets contain. Here, the criticality of transitions is estimated using measures we define for these experiments. The results show that, with some exceptions, the node contraction algorithm found a large proportion of the most critical cut sets in two orders of magnitude less time than did exhaustive enumeration. While further experiments are needed, these preliminary investigations suggest that minimal s-t cut set analysis can effectively identify critical transitions in large, complex Markov chain graphs as might be encountered in real-world problems.

1) Experimenal Design: To perform these experiments, four Markov chain models were selected, each consisting of 40 or 50 states, from [27-29] and single time-period TPMs were generated using [30]. All four problems were originally ergodic chains, which were suitably modified to be absorbing chains. Though the matrices were sparse, these problems were large and complex, with a very sizable number of minimal s-t cut sets ($> 4\times10^8$ for the largest) between the *Initial* and absorbing states. (See Table 2.) In contrast to the grid system model, minimal s-t cut sets for these problems consisted of multiple state transitions, which could correspond to combinations of circumstances that impact system performance. In [5], a complete description of these four Markov chain problems is provided, which must be omitted here due to space limitations.

TABLE II. Comparison of minimal s-t cut sets generated by the enumeration algorithm of [26] and by the node contraction algorithm. At 10,000 repetitions the node contraction generated 77.2 % (variance 555.2) of the top 100 ranked cut sets in 0.14 % of the time for Sorts A-C. At 100,000 repetitions, node contraction generated 91.4 % (variance 432.0) of the cut sets found by enumeration in 1.3 % of the time.

		Minimal s-t cut set enumeration		**Proportion (in %) of 100 top-ranked minimal s-t cut sets ranked by criteria A, B that were found by the node contraction algorithm**							
Number	Order	Number of cut sets	Time in hours	After 10,000 repetitions				After 100,000 repetitions			
				Time	Sort A	Sort B	Sort C	Time	Sort A	Sort B	Sort C
1	50	530,432	332 s	640 s	80	100	96	---	---	---	---
2	50	28,230,288	21.6	171 s	93	98	65	1710 s	99	100	99
3	50	27,242,634	36.0	218 s	67	100	100	2288 s	88	100	100
4	40	422,060,801	193.6	106 s	30	80	62	1051 s	37	100	100

To provide a baseline measure for the number of minimal s-t cut sets in these Markov chain graphs, the minimal s-t cut set enumeration algorithm of [26] was implemented, which lists all cut sets. To determine which minimal s-t cut sets were most critical, ranking criteria were selected based on the idea that the most critical cut sets will have a small number of state transitions. We chose this basis, because fewer transitions represent smaller combinations of circumstances that are more likely to occur and thus more likely to impact a system. (Note: this intuition is supported in the case of undirected graphs in [31] by the finding that small cut sets are more likely to disconnect undirected graphs, if edges of the graph that fail independently with a known probability.) We used this basis to choose 3 ranking criteria. The first criterion, Sort A, ranks minimal s-t cut sets by the fewest number of edges as the primary sorting criterion and lowest total transition probability of edges as the secondary criterion. The second, Sort B, uses only the lowest total transition probability of edges in the cut set as a sorting criterion (which also tends to rank cut sets with fewer transitions higher). Hence, Sorts A and B are likely to identify minimal s-t cut sets in which smaller perturbations to the fewest number of state transitions are likely to produce the largest changes. The third ranking criterion, Sort C, uses least number of edges as a primary sorting criterion and highest total transition probability of edges as a secondary criterion. Sort C identifies cut sets consisting of state transitions that are more likely to be taken and therefore, if perturbed, could affect system behavior.

2) Experimental Results: We applied the node contraction algorithm and the enumeration algorithm of [26] to the four TPMs, ranked the minimal cut sets produced by each using the ranking criteria described above, and compared the results to determine the proportion of most highly ranked cut sets that the node contraction algorithm could find. With the exception of Matrix 1, Table II shows that, with 100,000 repetitions, node contraction generated 91.4 % of the top 100 ranked cut sets that were generated by the enumeration algorithm for all four TPMs under all three sorting criteria. The contracton algorithm produced these results in 1.3 % of the time needed by enumeration. This amounts to a two-order of magnitude improvement in time. For instance, for Matrices 2 and 3, the algorithm was able to find almost all top 100 minimal s-t cut sets in a relatively small fraction of the number of hours required by the enumeration algorithm. For matrix 4, the node contraction algorithm could find all the top 100 under sort criteria B and C in about 15 minutes (as opposed to 156.1 hours through enumeration). Here, node contraction was successful despite the fact that Matrix 4 has over 4×10^8 minimal s-t cut sets. However, in Matrix 4, the algorithm found only 37 of 100 high ranked minimal s-t cut sets under Sort A. Also, for Matrix 1, Table II shows that the node contraction algorithm

had to run longer than the enumeration algorithm, before it began to produce a large percentage of highly-ranked cut sets. This difference in performance may also be attributable in part to topological characteristics such as vertices (states) with large numbers of edges (state transitions), which increases vertex interconnectivity and impedes the contraction process. This exception suggests that in some cases where TPMs are small, it may be more efficient to enumerate cut sets than to generate them probabilistically. Despite these exceptions, the data shows that the node contraction algorithm can be used to find a high proportion of minimal s-t cut sets representing combinations of critical state transitions in larger Markov chains within reasonable time limits.

VI. Using a Markov chain component to Support Monitoring of a distributed system

Using the grid system example, it is possible to envision a monitoring system organized as a real-time feedback control loop, in which a Markov chain analysis component with the capabilities described above runs as an on-demand background process. Such a Markov component might interact with a decision module and lower-level sensors as outlined below and depicted in Fig. 7.

To provide data for the Markov component, embedded sensors first record activity in the grid system about events such as negotiated service agreements and task completions, and forward the data to a sensor monitor. The monitor maps this sensor data to state transitions, and periodically sends this information to a TPM set computation module. The module uses this information to re-compute transition probabilities and update a TPM set for a piecewise homogenous representation, which is maintained to allow access by both the Markov component and decision module.

The Markov component may be activated either on a regular basis by the decision module or externally when design changes and other significant events occur. The Markov component maintains an updatable DTMC graph representation of the grid system. When activated, the Markov component obtains the most recent TPM information from the TPM set computation module. The Markov component then computes minimal s-t cut sets between the *Initial* and *Task Completed* states, and identifies critical state transitions and trap states. It also calculates threshold values for critical transitions and trap states, which when exceeded, indicate impending performance collapses (as in Figs. 4 and 6). The Markov component sends the results of its computations to the decision module.

The decision module also periodically obtains updated TPM information from the TPM computation module and matches this information to the critical transitions and trap states that were supplied by the Markov component. If, over time, the decision module observes a downward trend in the value of transition probabilities for critical transitions that exceeds a threshold, or a similar rise in the self-transition frequency of a trap state, it takes corrective action or notifies affected parties.

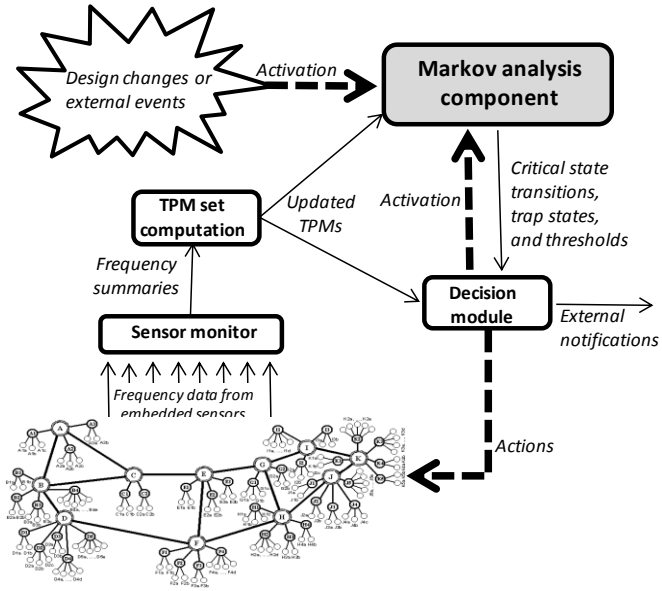


Figure 7. Use of a Markov chain analysis component in monitoring a distributed system. Thick solid arrows indicate control, while thin dashed arrows show data flow.

For example, suppose that over time the decision module receives updated TPM information which shows that the probability of self-transition to *Discovering* is increasing. This trend may reflect a partial failure of service discovery directories, which is prolonging discovery activity and causing tasks to stall in the *Discovering* state. As the trend worsens and the self-transition probability reaches a threshold, the decision module may initiate diagnostic procedures targeted to the service discovery function and notify a human operator. Note that the responsiveness of the self-management component is not impaired by the background Markov process, which itself does not operate in real time.

VII. Conclusions and Future Work

This paper has shown that the identification of minimal s-t cut sets on paths from the initial to the absorbing state can be used to perturb a piecewise homogenous Discrete Time Markov chain model of a dynamic distributed system in order to accurately and efficiently predict performance degradations. The paper has discussed how a probabilistic algorithm can be used for this purpose and shown that it is effective in analyzing a simulated grid computing system. Experimental results were provided that show the potential of this algorithm for efficiently analyzing large, complex problems and finding sets of related critical transitions that represent complicated circumstances. The perturbation of a piece-wise homogenous discrete Markov chain using minimal s-t cut set analysis has not been studied before for as a practical means for analysis of distributed system

dynamics. We believe that this combination of capabilities can potentially provide the basis for an operational tool for analyzing distributed system behavior and predicting performance degradations.

To further test the approach described here, we intend to carry out experiments on other kinds of distributed systems, such as computational clouds. As part of this work, we plan to extend the analysis to include minimal s-t separating sets, in which combinations of multiple trap states and state transitions represent circumstances that may affect system performance.

An important direction for future work is the investigation of other methods that are more effective in finding minimal s-t cut sets in directed graphs and can be used on larger, more complex problems. For instance, there are alternative approaches to probabilistic node contraction, such as [31] which could be examined. Another possible method involves use of maximum-flow algorithms [21-23] to find s-t cut sets that identify critical transitions. These algorithms find s-t cut sets on the basis of maximum flow and minimum capacity. Potentially, maximum-flow algorithms could be adapted to find and rank cut sets on the basis of their nearness to maximum flow and minimum capacity, rather than criteria described here. To enable such rankings, the work of [32, 33] could be used. Finally, we are also exploring the use of eigensystem analysis of Markov chain models as a means to predict performance degradation [5].

REFERENCES

[1] K. Mills, and C. Dabrowski. "Investigating Global Behavior in Computing Grids," LNCS, vol. 4124. pp. 120-136. Springer, 2006.

[2] C. Dabrowski, and F. Hunt. "Using Markov Chain Analysis to Study Dynamic Behavior in Large-Scale Grid Systems," Seventh Australasian Symp. on Grid Computing and e-Research, 2009.

[3] D. Rosenberg, and E. Solan. Vielle N. "Approximating A Sequence of Observations by a Simple Process," The Annals of Statistics, vol. 32, no. 6, pp. 2742-2775, 2004.

[4] S. Tsukiyama, I. Shirakawa, H. Ozaki, and H. Ariyoshi. "An Algorithm to Enumerate All Cut Sets of a Graph in Linear Time per Cutset," Jour. of the ACM. vol. 27, no.4, pp. 619-632, 1980.

[5] C. Dabrowski, F. Hunt, and K. Morrison. Improving the Efficiency of Markov Chain Analysis of Complex Distributed Systems. Draft NIST Interagency Report, 2010.

[6] J. Wu, and F. Deng. "Finite Horizon Optimal Control of Networked Control Systems with Markov Delays." Proc. of the Sixth World Congress on Intelligent Control and Automation. pp. 4513-4517, 2006.

[7] D. Feng, D. Wencai, and L. Zhi. "New Smith Predictor and Nonlinear Control for Networked Control Systems," Proc. of the International MultiConference of Engineers and Computer Scientists, *2009* (Volume II), pp. 1148-1153., 2009.

[8] P. Schweitzer. "Perturbation Theory and Finite Markov Chains." Jour. of Applied Probability, vol. 5, no. 2, pp. 401-413, 1968.

[9] F. Delebecque. "A Reduction Process for Perturbed Markov Chains," SIAM Jour. of Applied Mathematics, vol. 43, pp. 325-350, 1983.

[10] R. Hassin, and M. Haviv. "Mean Passage Times and Nearly Uncoupled Markov Chains," SIAM Jour. of Discrete Mathematics, vol. 5, no. 3, pp. 386-397, 1992.

[11] C. Meyer. "Stochastic Complementation, Uncoupling Markov Chains, and the Theory of Nearly Reducible Systems," SIAM Review, vol. 31, no. 2, pp. 240-272, 1989.

[12] W. Stewart, and M. Dekker. Numerical Solution of Markov Chains, Princeton: Princeton University Press, 1994.

[13] Y. Ho, and S. L. "Extensions of infinitesimal perturbation analysis," IEEE Trans. on Automation Control, vol. AC-33, no. 5, pp. 427-438, 1988.

[14] R. Suri. "Perturbation Analysis: The State of the Art and Research Issues Explained via the GI/G/l Queue," *Proc. of the IEEE*, vol. 77, no. 1, pp. 114-138, 1989.

[15] X. Cao, and J. Zhang. "Event-Based Optimization of Markov Systems," IEEE Trans. on Automatic Control, vol. 53, no. 4, pp. 1076-1082, 2008.

[16] M. Benzi, and M. Tuma. "A parallel solver for large-scale Markov chains," Applied Numerical Mathematics, vol. 41, pp. 135-153, 2002.

[17] A. Gambin, P. Kryzanowski, and P. Pokarowski. "Aggregation Algorithms for Perturbed Markov Chains with Applications to Network Modeling," *SIAM* Jour. of Scientific Computation, vol. 31, no.1, pp. 45-77, 2008.

[18] G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. "Markovian Analysis of Large Finite State Machines," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 12, 1996.

[19] E. Solan, and N. Vielle. "Perturbed Markov Chains," Jour. of Appl. Prob. vol. 40, pp. 07-122, 2003.

[20] Z. Tang, and J. Dugan. "Minimal cut set/sequence generation for dynamic fault trees," Proc. of the 2004 Annual Symp. on Reliability and Maintainability, pp. 207- 213, 2004.

[21] L. Ford, and D. Fulkerson. Flows in Networks. Princeton: Princeton University Press, 1962.

[22] S. Even. Graph Algorithms. Rockville, MD: Computer Science Press, 1979.

[23] A. Goldberg, and R. Tarjan. "A New Approach to the Maximum-Flow Problem," Jour. of the ACM. vol. 35, no. 4, pp. 921-940, October 1988.

[24] J. Hayakawa, S. Tsukiyama, and H. Ariyoshi. "Generation of Minimal Separating Sets of Graphs," IEICE Transaction Fundamentals, vol.E82-A, no. 5, pp. 775-783, 1999.

[25] D. Karger, and C. Stein. "A New Approach to the Minimum Cut Problem," Jour. of the ACM, vol. 43, pp. 601-640, 1996.

[26] J. Provan, and D. Shier. "A Paradigm for Listing (s,t)-cuts in Graphs," Algorithmica., vol. 15, pp. 351-372, 1996.

[27] A. Boyarksy. "A matrix method for estimating the Liapunov exponent of one-dimensional systems," Jour. of Statistical Physics, vol. 50, no. 1-2, pp. 213-229, 1988.

[28] RW136: Markov Chain Transition Probability Matrix, National Institute of Standards and Technology, http://math.nist.gov/MatrixMarket/data/NEP/mvmrwk/rw136.html, 2009.

[29] R. Jensen, and E. Jessup. "Statistical Properties of the Circle Map," Jour. of Statistical Physics, vol. 43, no. 1-2, pp. 369-389, 1986.

[30] F. Hunt. "A Monte Carlo Approach To The Approximation of Invariant Measures," Random and Computational Dynamics, vol. 2, no. 1, pp. 111-112, 1994.

[31] D. Karger. "A Randomized Fully Polynomial Time Approximation Scheme for the All-Terminal Network Reliability Problem," SIAM Review, vol. 43, no. 3, pp. 499-522, 2001.

[32] N. Curet, J. DeVinney, and M. Gaston. "An Efficient Network Flow Code for Finding all Minimum Cost s-t Cutsets," Computers and Operations Research, vol. 29, pp. 205-219, 2000.

[33] A. Balcioglu, and K. Wood. "Enumerating Near-Min s-t Cuts," In Network Interdiction and Stochastic Integer Programming, ed. D. Woodruff, Kluwer Academic Publishers, pp. 21-49, 2003.